Journal of Computational and Applied Mathematics 28 (1989) 129–143
North-Holland

Efficiency results for Gaussian elimination on the iPSC/2 hypercube

Lutgarde BEERNAERT and Dirk ROOSE
Department of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200 A, B-3030 Heverlee, Belgium

Received 5 August 1988

Abstract: We compare two methods for solving banded linear systems on a hypercube multiprocessor. Both methods are based on Gaussian elimination. The differences in the methods are due to different allocation schemes to distribute the data among the nodes. We implemented both methods on the Intel iPSC/2 hypercube. Timing results and efficiency results obtained on this multiprocessor are discussed.

Keywords: Parallel algorithms, hypercube, band matrices, Gaussian elimination.

1. Introduction

We want to solve banded linear systems, i.e., we want to know X in

$$AX = B,$$

with A an $n \times n$-matrix with half bandwidth ν (i.e., A has $\nu - 1$ diagonals above and under the main diagonal), and with B and X $n \times m$-matrices. We use Gaussian elimination to decompose A into $L \cdot U$ with L a lower triangular matrix with diagonal elements 1 and U an upper triangular matrix. If pivoting is not incorporated, L and U are band matrices with bandwidth ν. If row pivoting is taken into account, the bandwidth of L remains ν, but the bandwidth of U becomes $2\nu - 1$. After the LU-decomposition the triangular systems $LY = B$ and $UX = Y$ are solved.

For the allocation schemes we use, the processors are ordered in a square grid. This can be represented by a square matrix D (size $\kappa \times \kappa$), see Fig. 1. For a hypercube this implies that the dimension of the cube must be even.

2. Allocation schemes

For the first allocation scheme (allocation scheme of Fox) [3,4], the matrix A is, starting in the top left-hand corner, covered with disjunct copies of the matrix D. Then each element within the band is allocated to the processor that covers it. See Fig. 2 for an example.

For the allocation scheme of Saad and Schultz [5] square blocks of elements are allocated to a processor. In order to obtain these square blocks some of the zero elements out of the band must

$$D = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix}$$

Fig. 1. Matrix D that represents the grid of processors.

1	2	3	4	1	2										
5	6	7	8	5	6	7									
9	10	11	12	9	10	11	12								
13	14	15	16	13	14	15	16	13							
1	2	3	4	1	2	3	4	1	2						
5	6	7	8	5	6	7	8	5	6	7					
	10	11	12	9	10	11	12	9	10	11	12				
		15	16	13	14	15	16	13	14	15	16	13			
			4	1	2	3	4	1	2	3	4	1	2		
				5	6	7	8	5	6	7	8	5	6	7	
					10	11	12	9	10	11	12	9	10	11	12
						15	16	13	14	15	16	13	14	15	16
							4	1	2	3	4	1	2	3	4
								5	6	7	8	5	6	7	8
									10	11	12	9	10	11	12
										15	16	13	14	15	16
											4	1	2	3	4
												5	6	7	8

Fig. 2. Allocation of a matrix with half bandwidth 6 on a 4×4-grid of processors, following the method of Fox.

be used. The dimension of the blocks depends on the bandwidth of the matrix A and on the number of processors available. It is given by the formula [1]

$$\lambda = \left\lceil \frac{\nu - 1}{\kappa - 1} \right\rceil;$$

λ is such that each processor of the first row of D appears with just 1 block on the first row of A. See Fig. 3 for an example. The figures printed in bold represent zeros that are taken into account in order to obtain square blocks.

1	1	2	2	3	3	4	4								
1	1	2	2	3	3	4	4								
5	5	6	6	7	7	8	8	5	5						
5	5	6	6	7	7	8	8	5	5						
9	9	10	10	11	11	12	12	9	9	10	10				
9	9	10	10	11	11	12	12	9	9	10	10				
13	13	14	14	15	15	16	16	13	13	14	14	15	15		
13	13	14	14	15	15	16	16	13	13	14	14	15	15		
		2	2	3	3	4	4	1	1	2	2	3	3	4	4
		2	2	3	3	4	4	1	1	2	2	3	3	4	4
				7	7	8	8	5	5	6	6	7	7	8	8
				7	7	8	8	5	5	6	6	7	7	8	8
						12	12	9	9	10	10	11	11	12	12
						12	12	9	9	10	10	11	11	12	12
								13	13	14	14	15	15	16	16
								13	13	14	14	15	15	16	16
										2	2	3	3	4	4
										2	2	3	3	4	4

Fig. 3. Allocation of a matrix with half bandwidth 6 on a 4×4-grid of processors, following the method of Saad and Schultz.

1	2	3	4	1
5	6	7	8	5
9	10	11	12	9
13	14	15	16	13
1	2	3	4	1
5	6	7	8	5
9	10	11	12	9
13	14	15	16	13
1	2	3	4	1
5	6	7	8	5
9	10	11	12	9
13	14	15	16	13
1	2	3	4	1
5	6	7	8	5
9	10	11	12	9
13	14	15	16	13
1	2	3	4	1
5	6	7	8	5

1	2	3	4	1
1	2	3	4	1
5	6	7	8	5
5	6	7	8	5
9	10	11	12	9
9	10	11	12	9
13	14	15	16	13
13	14	15	16	13
1	2	3	4	1
1	2	3	4	1
5	6	7	8	5
5	6	7	8	5
9	10	11	12	9
9	10	11	12	9
13	14	15	16	13
13	14	15	16	13
1	2	3	4	1
1	2	3	4	1

Fig. 4. Allocation of the matrix B following the allocation scheme of Fox and the allocation scheme of Saad and Schultz.

Later on it will become clear that the value λ is crucial for this allocation scheme. For the allocation scheme of Fox we define

$$\mu = \left\lceil \frac{\nu}{\kappa} \right\rceil;$$

μ^2 is an upper bound for the number of elements a processor has in a window (see next section). λ and μ can be equal, if $\nu \approx \kappa$. However, if ν is much larger than κ, λ is much larger than μ.

The allocation of the matrix B is such that a row of B is allocated to the same row of processors as the corresponding row of A. For the allocation scheme of Fox we map the matrix D of processors on the matrix B, again starting in the top left-hand corner. If B has fewer columns than D, not all processors have elements of B. This does not imply that they would not be active during the solving phase. We define

$$M = \left\lceil \frac{m}{\kappa} \right\rceil.$$

For the allocation scheme of Saad and Schultz λ successive rows are allocated to the same row of processors. For reasons of load-balancing it is better to use $\lambda \times 1$ blocks, instead of $\lambda \times \lambda$ blocks. The value M has the same meaning for this allocation scheme. See Fig. 4 for examples of the allocation of the matrix B to the processors.

3. LU-decomposition

For the allocation scheme of Fox in the first step of the LU-decomposition without pivoting only the square submatrix of dimension $\nu \times \nu$ starting in the left-hand upper corner of A must be considered. This submatrix is called a window. In each of the following steps the window is shifted one place along the main diagonal to the right-hand bottom corner. For the allocation scheme of Saad and Schultz windows of size $\kappa\lambda \times \kappa\lambda$ are considered. These windows consist of κ^2 $\lambda \times \lambda$-blocks, one allocated to each processor. As long as the pivot row is allocated to the

same row of processors (during λ consecutive steps) the same window is used. Then the window is shifted λ places to the right-hand bottom corner.

This gives the following algorithm:

For $k = 1$ to $n - 1$ do:

1. Broadcast the pivot row vertically: the processors containing part of the pivot row send it to the processors in the same grid-column.
2. Compute the multipliers and broadcast them horizontally: the processors containing part of the first column of the current window compute the multipliers $l_{ik} = a_{ik}/a_{kk}$ and send these to the processors in the same grid-row.
3. Compute the new values for the elements a_{ij}: all processors work in parallel.

Since each processor has at most μ^2, respectively λ^2, elements in a window, we find that this algorithm requires $2\mu^2 + \mu$, respectively $2\lambda^2 + \lambda$ numerical operations and 2 messages of μ, respectively λ, elements. Since $\mu \leqslant \lambda$ the allocation scheme of Fox yields an algorithm that performs better with respect to both computational work and communication.

If row pivoting is used, the windows must be enlarged: for the allocation scheme of Fox to $\nu \times (2\nu - 1)$ and for the allocation scheme of Saad and Schultz to $\kappa\lambda \times (2\kappa - 1)\lambda$.

We then obtain the algorithm:

For $k = 1$ to $n - 1$ do:

1. Determine the pivot row: the processors containing part of the first column of the current window find the maximum of $|a_{ik}|$ for $k \leqslant i \leqslant \min(k + \nu - 1, n)$.
2. Interchange the pivot row and row k: the processors containing part of the pivot row and the processors containing part of row k interchange these parts. The interchanges must be stored for using them later while solving the system $Ly = b$. The interchanging must be performed explicitly, otherwise the advantage of using a window is lost. The multipliers of row k must not be interchanged in order to maintain the band structure of the matrix L.
3. Broadcast the pivot row vertically: the processors containing part of the pivot row send it to the processors in the same grid-column.
4. Compute the multipliers and broadcast them horizontally: the processors containing part of the first column of the current window compute the multipliers $l_{ik} = a_{ik}/a_{kk}$ and send these to the processors in the same grid-row.
5. Compute the new values for the elements a_{ij}: all processors work in parallel.

Bearing in mind that parts 2 and 3 can be executed together, the additional work due to pivoting is, for the allocation scheme of Fox, per step, $\mu - 1$ comparisons in the processors, $\log_2 \kappa$ comparisons between different processors (these comparisons require communication), 1 more message of 2μ elements for sending row k to the processors that contain the pivot row. Notice that now the messages in part 3 have length 2μ instead of μ, if pivoting is not considered. For the allocation scheme of Saad and Schultz these numbers are: $\lambda - 1$ comparisons in the processors, $\log_2 \kappa$ comparisons between processors, 1 message of 2λ elements and messages of length 2λ instead of λ. The algorithm of Fox performs better with respect to both the number of comparisons and the communication.

One may remark that information (pivot row and column of multipliers) could be pipelined instead of broadcasted. This would imply a longer execution time for each pass of the for-loop, if these passes were executed separately. However, if the different passes of the for-loop are also pipelined, the total execution time of the algorithm will decrease. This improvement is not possible if row-pivoting is considered, since then at least column k has to be updated in pass $k-1$ before pass k can start and the new pivot row has to be updated before it can be distributed. Thus, pivoting induces a synchronization of the processors after each pass of the for-loop, which implies that the different passes cannot be pipelined. Since we are mainly interested in the case of pivoting, we select an algorithm suited for that case, and we accept that the adapted version for the case of no-pivoting is not optimal.

4. Forward and backward substitution

4.1. Without pivoting

In this case, the algorithms for the forward and the backward substitution are very similar. Therefore only forward substitution is treated here.

4.1.1. Allocation scheme of Fox

for $k = 1(1)n$

{ distribute column k of L
{ solve for row k of Y
{ distribute row k of Y vertically
{ adapt elements of B

4.1.2. Allocation scheme of Saad and Schultz

for $k = 1(\lambda)n$

{ distribute columns $k, \ldots, k+\lambda-1$ of L
{ solve for rows $k, \ldots, k+\lambda-1$ of Y
{ distribute rows $k, \ldots, k+\lambda-1$ of Y vertically
{ adapt elements of B

4.1.3. Discussion

Comparing the algorithms shows that for the allocation scheme of Saad and Schultz a block version of the algorithm of the allocation scheme of Fox is used. The algorithm following the allocation scheme of Saad and Schultz requires more computational work, and also more elements are transported. But the messages used are longer. Especially on machines with a large startup time for communication, it is important to be able to send fewer but longer messages. That is why in this case the algorithm following the allocation scheme of Saad and Schultz performs better than the one following the allocation scheme of Fox.

4.2. With pivoting

4.2.1. Allocation scheme of Fox

for $k = 1(1)n$

{ distribute column k of L
interchange row k and row pivot $[k]$ of B
solve for row k of Y
distribute row k of Y vertically
adapt elements of B

4.2.2. Allocation scheme of Saad and Schultz

for $k = 1(1)n$

{ if $(k-1) \bmod \lambda = 0$
  distribute columns $k, \ldots, k+\lambda-1$ of L
interchange row k and row pivot $[k]$ of B
solve for row k of Y
distribute row k of Y vertically
adapt elements of B

4.2.3. Discussion

Since in each step two rows of B must be interchanged, the advantage of working with blocks (Saad and Schultz) is partly lost. But we still distribute λ columns of L in one message. So the allocation scheme of Saad and Schultz is for this case also better than the allocation scheme of Fox.

For the backward substitution the same algorithm as in the case of no pivoting can be used. Only the bigger bandwidth must be taken into account. The results of Section 4.1.3 remain valid.

5. Theoretical efficiency

5.1. LU-decomposition

LU-decomposition without pivoting of an $n \times n$-matrix A with half bandwidth ν requires approximately $2\nu^2 n$ numerical operations. In the algorithm described above each processor executes approximately $2\mu^2 n$ numerical operations for the allocation scheme of Fox and $2\lambda^2 n$ numerical operations for the allocation scheme of Saad and Schultz.

We can define the *maximal speedup* as the number of necessary operations divided by the number of operations executed per processor. This yields for the allocation schemes:

$$\text{Fox}: \frac{2\nu^2 n}{2\mu^2 n} = \frac{\nu^2}{\mu^2} = \frac{\nu^2}{\left\lceil \frac{\nu}{\kappa} \right\rceil^2} \approx \kappa^2;$$

$$\text{Saad and Schultz}: \frac{2\nu^2 n}{2\lambda^2 n} = \frac{\nu^2}{\lambda^2} = \frac{\nu^2}{\left\lceil \frac{\nu-1}{\kappa-1} \right\rceil^2} \approx \frac{\nu^2}{(\nu-1)^2}(\kappa-1)^2.$$

From this last formula could be deduced that for small values of ν the allocation scheme of Saad and Schultz allows theoretically a higher speedup than the scheme of Fox. But in practice, communication will prohibit a high speedup, especially if ν is small. If ν is large, the maximal speedup is approximately $(\kappa-1)^2$. For the maximal efficiency this yields:

$$\frac{\kappa^2}{\kappa^2}=1 \text{ (Fox)} \quad \text{and} \quad \frac{(\kappa-1)^2}{\kappa^2} \text{ (Saad and Schultz).}$$

For the allocation scheme of Fox speedup κ^2 is attainable, because no useless operations are executed. For the allocation scheme of Saad and Schultz the best possible speedup attainable is $(\kappa-1)^2$. This is especially dramatic for $\kappa=2$. Then no speedup can be attained.

If pivoting is incorporated, the number of numerical operations is higher, but the maximal speedup is the same.

5.2. Solving the systems

Forward and backward substitution without pivoting of an $n \times n$-lower or -upper triangular band matrix with bandwidth ν and m right-hand sides, requires approximately $2\nu mn$ numerical operations. In the algorithms described above each processor executes approximately $2\mu nM$ numerical operations for the allocation scheme of Fox and $2\lambda nM$ numerical operations for the allocation scheme of Saad and Schultz.

The maximal speedup is then:

$$\text{Fox}: \frac{2\nu nm}{2\mu nM}=\frac{\nu m}{\mu M}=\frac{\nu m}{\left\lceil\frac{\nu}{\kappa}\right\rceil\left\lceil\frac{m}{\kappa}\right\rceil} \approx \kappa^2;$$

$$\text{Saad and Schultz}: \frac{2\nu nm}{2\lambda nM}=\frac{\nu m}{\lambda M}=\frac{\nu m}{\left\lceil\frac{\nu-1}{\kappa-1}\right\rceil\left\lceil\frac{m}{\kappa}\right\rceil} \approx \frac{\nu}{\nu-1}(\kappa-1)\kappa \approx (\kappa-1)\kappa.$$

For the maximal efficiency this yields:

$$\frac{\kappa^2}{\kappa^2}=1 \text{ (Fox)} \quad \text{and} \quad \frac{(\kappa-1)\kappa}{\kappa^2}=\frac{\kappa-1}{\kappa} \text{ (Saad and Schultz).}$$

The numbers for maximal speedup and efficiency are higher for the allocation scheme of Fox than for the allocation scheme of Saad and Schultz. But these numbers do not take into account any communication. The kind of communication in the algorithm following the allocation scheme of Fox (a large number of short messages) prevents it more from approaching the maximal speedup than it does in the algorithm following the allocation scheme of Saad and Schultz (fewer but longer messages).

If pivoting is incorporated, the number of numerical operations for the backward substitution is higher, but the maximal speedup is the same: κ^2 for the allocation scheme of Fox and $(\kappa-1)\kappa$ for the allocation scheme of Saad and Schultz. For the forward substitution the number of numerical operations remains the same as in the case of no-pivoting, but the communication increases, which will have a negative influence on the speedup.

6. Timing results

6.1. Introduction

In the following we define the speedup S_p as the time it takes to execute a program on 1 processor divided by the time it takes to execute the same program on p processors, i.e.,

$$S_p = \frac{T_1}{T_p}.$$

Hence, in T_1 also the organization overhead of the parallel algorithm is incorporated. The efficiency E_p is given by

$$E_p = \frac{S_p}{p}.$$

The allocation scheme of Saad and Schultz does not exist for 1 processor. In deriving the formula for λ it is assumed implicitly that κ (see Section 2) is not equal to 1. In order to compute the speedup for this allocation scheme, we used an implementation that ensured that on

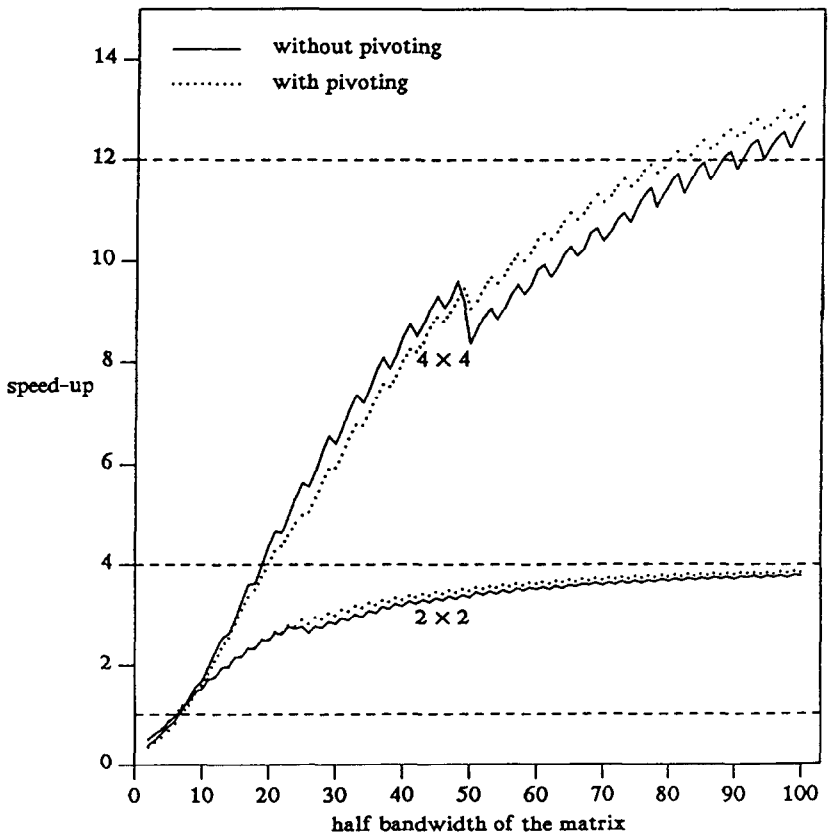

Fig. 5. LU-decomposition with and without pivoting: speedup as a function of the half bandwidth for the allocation scheme of Fox. (Dimension of the matrix: 500; speedup: $S_p = T_1/T_p$).

1 processor no useless operations or communications were executed, but the same data structure was used as in the program executed on 4 or 16 processors.

6.2. LU-decomposition

6.2.1. Allocation scheme of Fox

From the description of the algorithm, it easily follows that the efficiency is completely determined by the bandwidth. The dimension of the matrix has only a minor influence in the treatment of the end of the band matrix. In all our computations we used random matrices of dimension 500. In Fig. 5 the speedup is shown as a function of the half bandwidth ν, from 2 up to 100. Some remarks are:

- All graphs have a saw tooth behaviour. This is due to the fact that the time of the algorithm depends on μ rather than on ν. Several values of ν give rise to the same value of μ if $\kappa > 1$, while for $\kappa = 1$ each increase in ν induces the same increase in μ.
- For matrices with a narrow bandwidth, overhead due to communication is so high that LU-decomposition takes less time on 1 processor than on 4 or 16 processors.
- With 4 processors, a speedup of more than 3.6, i.e., an efficiency of over 90%, is reached for large bandwidths ($\nu \geqslant 90$). Asymptotically the maximal speedup can be reached.
- With 16 processors a speedup of about 13 is obtained, which yields an efficiency of 80%. In this case still larger bandwidths are necessary for approaching the maximal speedup.

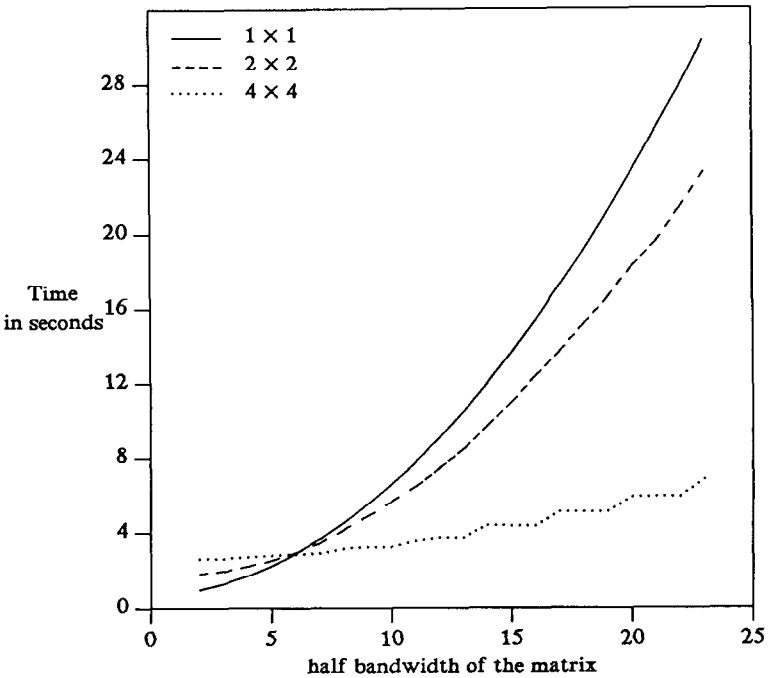

Fig. 6. LU-decomposition with pivoting: computing time as a function of the bandwidth for a matrix of dimension 500 (allocation scheme of Saad and Schultz).

- In the 4×4-graph for the case without pivoting a remarkable decrease in speedup appears if the half bandwidth ν increases from 48 to 49. This is due to the communication system on the Intel iPSC/2. Messages shorter than 100 bytes are sent immediately to the destination processor and stored there in one of the system buffers available for short messages. For messages longer than 100 bytes first a path must be established and a buffer of the appropriate length must be applied for, and then the message can be sent. This double system of communications induced a big increase in time if the message length becomes larger than 100 bytes.

  If $\nu = 48$ and $\kappa = 4$, then $\mu = 12$, and hence the length of the messages is 96 bytes. If $\kappa = 49$, then $\mu = 13$. Now some of the messages have length 104 bytes, others have length 96 bytes, and the messages of length 104 take more time.

  This phenomenon also appears in the graph for $\kappa = 2$, if ν increases from 24 to 25, but less pronouncedly, because the ratio of communication time to computation time is smaller. If pivoting is considered, both shorter and larger messages are sent. Therefore the increase in the half bandwidth ν does not have such a big influence.

6.2.2. Allocation scheme of Saad and Schultz

The timing results of Fig. 6 show that for this allocation scheme working on 4 processors takes nearly the same time as working on 1 processor, so that hardly any speedup is obtained. On 16 processors a higher speedup is obtained, but efficiency remains very low (Fig. 7). Even with a large bandwidth, only a limited speedup can be obtained.

6.2.3. Comparison of the allocation schemes

From Fig. 8 it is clear that the allocation scheme of Fox performs best. LU-decomposition, both with and without pivoting is computed much faster. A higher speedup, and as a consequence, a higher efficiency are obtained.

6.3. Solving the triangular systems

In Fig. 9 we compare times for computing the backward substitution for one right-hand side with and without pivoting as a function of the bandwidth ν for the allocation scheme of Fox.

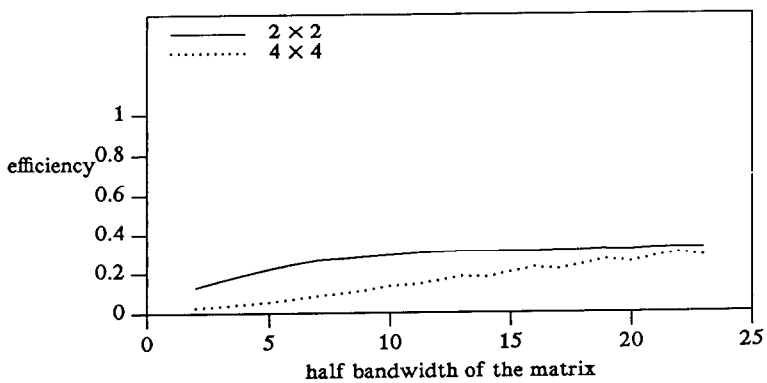

Fig. 7. LU-decomposition with pivoting: efficiency for a matrix of dimension 500 as a function of the bandwidth (allocation scheme of Saad and Schultz; efficiency: $E_p = S_p/p$).

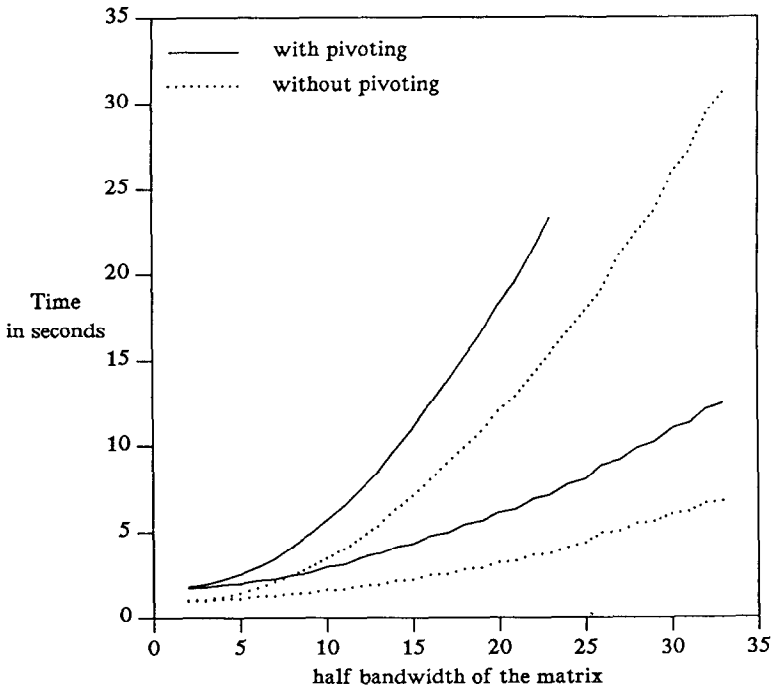

Fig. 8. LU-decomposition with and without pivoting: computing time as a function of the bandwidth for a matrix of dimension 500 on 16 processors (upper curves: Saad and Schultz; lower curves: Fox).

In the case without pivoting, the algorithm is fastest on 1 processor, except when the bandwidth becomes larger than 16. If pivoting is incorporated, the situation is slightly better, because the bandwidth of the matrix U is now $2\nu - 1$ instead of ν, so that the number of numerical operations is almost twice as high as in the case without pivoting.

All curves are nearly straight lines. In Fig. 9(b) the lines that give the times on 4 or 16 processors have a very small slope. This is because the communication dominates the computational work and all short messages take nearly the same time. In Fig. 9(a) the execution times increase faster because there is more computational work and the messages are twice as large for distributing the columns of U.

If the number of right-hand sides increases, the computational work increases faster than the communication, so that a better speedup can be obtained.

In Fig. 10 we compare forward (Fig. 10(a)) and backward (Fig. 10(b)) substitution for the allocation scheme of Saad and Schultz for 8 right-hand sides.

On 4 and 16 processors the forward substitution takes more time than the backward substitution, although each processor has twice as much computational work for the backward substitution than for the forward substitution, which is very clear if the computing times on one processor are compared. This proves that communication dominates the forward substitution and the short messages for interchanging two rows of Y are very time-consuming. On a computer system with a lower startup time for communication other results could be obtained.

In Fig. 11 we compare computing times for forward and backward substitution for both allocation schemes as a function of the number of right-hand sides. The half bandwidth of the matrix is 10. We first compare the allocation schemes.

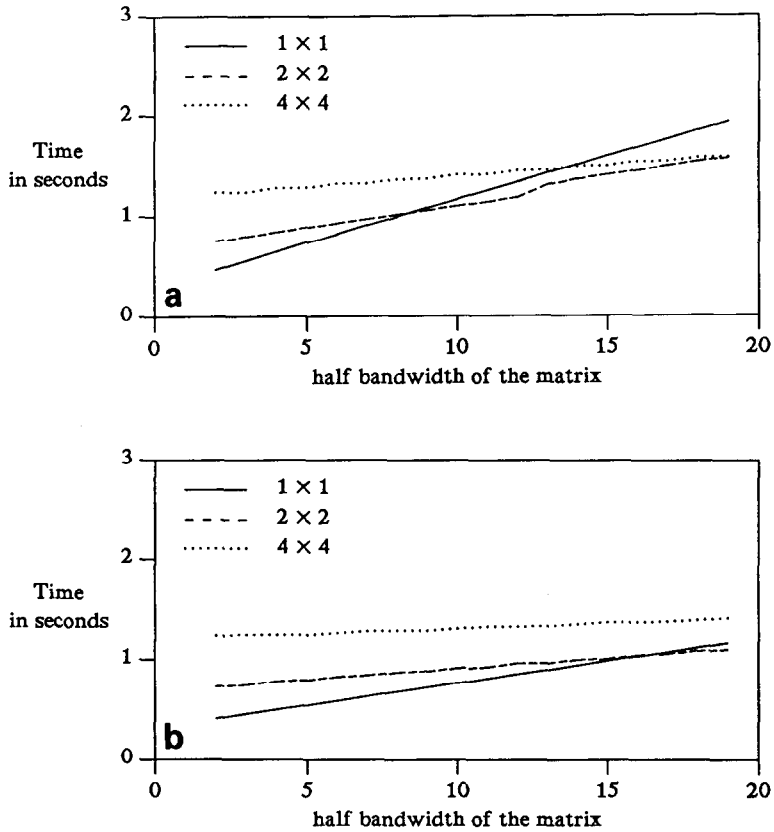

Fig. 9. Backward substitution with (a) and without (b) pivoting for the allocation scheme of Fox: computing time on 1, 4 and 16 processors as a function of the bandwidth. Number of right-hand sides: 1. Dimension of the matrix: 500.

- For a small number of right-hand sides, the allocation scheme of Saad and Schultz is to be preferred for both forward and backward substitution. This is due to the kind of communication. A lot of short messages take more time than fewer but longer messages, even if more elements are to be transported.
- For a large number of right-hand sides the allocation scheme of Fox becomes better for both forward and backward substitutions. Now the computational part becomes more important, and the algorithm following the allocation scheme of Fox requires fewer computations.

We now compare forward and backward substitution.

- For a small number of right-hand sides backward substitution is faster than forward substitution for both allocation schemes, although it requires more computational work. Here again communication dominates and the backward substitution requires less communication than the forward substitution.
- For a large number of right-hand sides forward substitution is faster for both allocation schemes, because now the computational part is dominant. Communication and computations can overlap.

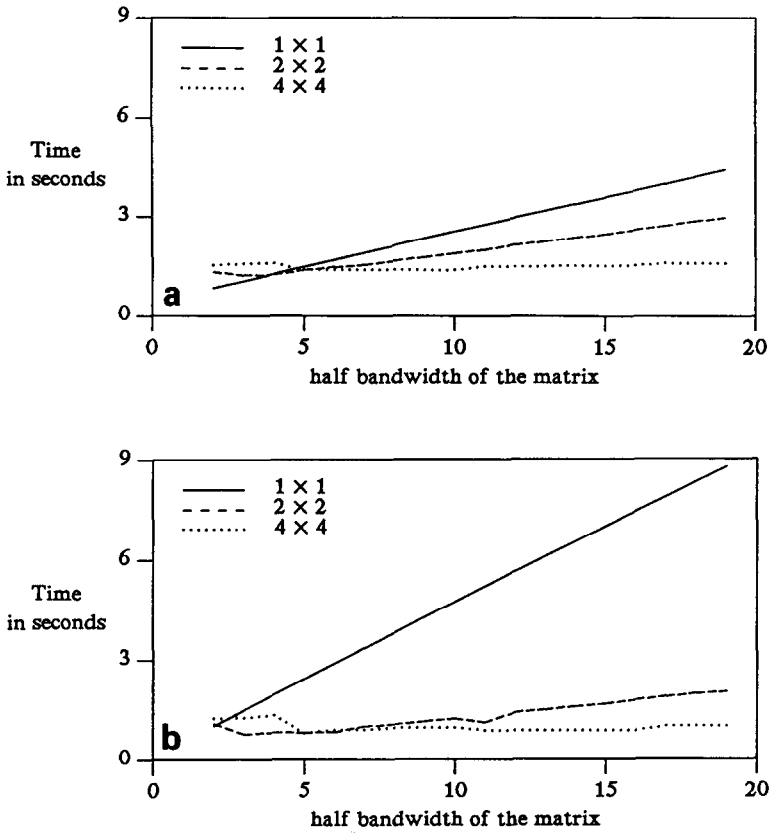

Fig. 10. Forward (a) and backward (b) substitution with pivoting for the allocation scheme of Saad and Schultz: computing time as a function of the bandwidth. Number of right-hand sides: 8. Dimension of the matrix: 500.

Here also a big increase in time appears in the graph of the forward substitution of both allocation schemes if the number of right-hand sides increases from 48 to 49. Again this is due to the system of communications of the Intel iPSC/2. For the backward substitution this phenomenon is less pronounced, because messages of several lengths are being sent.

7. Conclusions

We compared two methods for solving banded linear systems on a hypercube multiprocessor. The difference between the methods is due to different allocation schemes. Theoretically the method following the allocation scheme of Fox is better than the method following the allocation scheme of Saad and Schultz for the LU-decomposition. For the solution of the resulting triangular systems the method following the allocation scheme of Saad and Schultz is best. The timing results obtained on the Intel iPSC/2 multiprocessor agree with these theoretical results.

Since each of the two allocation schemes is best for part of the algorithm, the question arises whether there exists an optimal blocksize, between 1 and λ. LU-decomposition is the dominating part in the algorithm, except when the number of right-hand sides is very large or the same

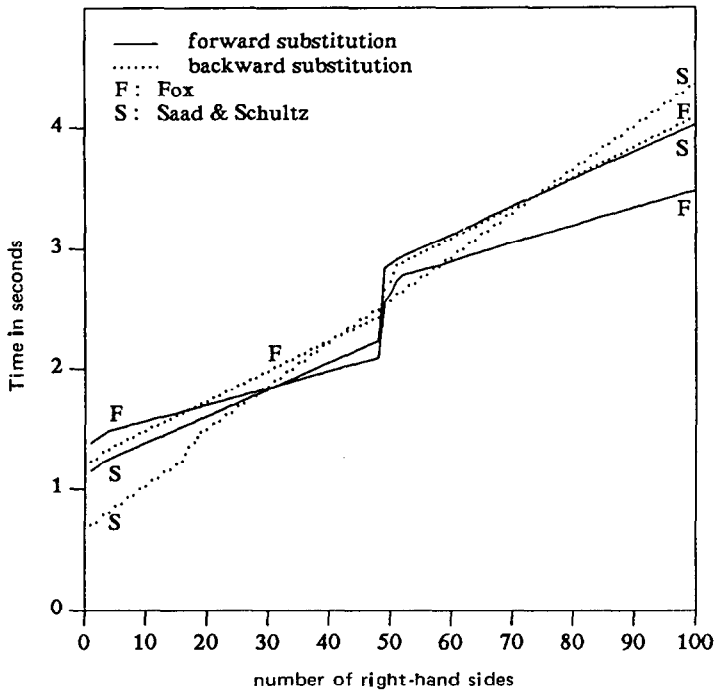

Fig. 11. Forward and backward substitution with pivoting: computing times on 16 processors as a function of the number of right-hand sides, half bandwidth: 10. Dimension of the matrix: 500.

matrix is used for a large number of successive sets of right-hand sides. Therefore it is best to use the optimal blocksize for the LU-decomposition. This optimal blocksize is 1 (allocation scheme of Fox), since, whatever other blocksize is chosen, some of the zero-entries of the matrix are considered as nonzeros, which implies that more computational work and more communication will be done.

For the allocation scheme of Fox a maximal speedup equal to the number of processors can be obtained if the problem is large enough, i.e., if the bandwidth of the matrix is large and if there are many right-hand sides. For the allocation scheme of Saad and Schultz this is not true. These maximal speedups are, however, not reached, because of the communication. In these algorithms communication consists of many short messages. On multiprocessors with a high startup time for communication, as the iPSC/2, the execution time is dominated by communication. However, these algorithms are well-suited for systems on which short messages can be communicated efficiently. This is shown in [2].

Acknowledgement

The authors would like to thank Kurt Wayenberg for his help with the implementation.

References

[1] L. Beernaert, D. Roose and K. Wayenberg, Gaussian elimination of banded linear systems with pivoting on a hypercube, Report TW 101, Department of Computer Science, Katholieke Universiteit Leuven, 1987.

[2] L. Beernaert, S. Van Praet, D. Roose and P. de Groen, Parallel Gaussian elimination, iPSC/2 versus a transputer network, in: P. Van Dooren, Ed., *Proceedings of the Nato Advanced Study Institute on Numerical Linear Algebra, Digital Signal Processing and Parallel Algorithms, Leuven. 1988*, to appear.

[3] G. Fox, LU-decomposition for banded matrices, Report Hm-99, California Institute of Technology, Pasadena, CA, 1984.

[4] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon and D. Walker, *Solving Problems on Concurrent Processors* (Prentice-Hall, Englewood Cliffs, NJ, 1988).

[5] Y. Saad and M. Schultz, Parallel direct methods for solving banded linear systems, *Linear Algebra Appl.* (1988) 623–650.